Design of an Integrated Environment for the Automated Analysis of Architectural Drawings

Philippe Dosch, Christian Ah-Soon, Gérald Masini, Gemma Sánchez, Karl Tombre

Design of an Integrated Environment for the Automated Analysis of Architectural Drawings[a]

Philippe Dosch, Christian Ah-Soon, Gérald Masini, Gemma Sánchez[b], Karl Tombre
LORIA–CNRS–INPL–INRIA–UHP
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{dosch, ahsoon, masini, gsanchez, tombre}@loria.fr
http://www.loria.fr/isa/

In this paper, we present the principles which have guided the design of our graphics recognition software environment. We show a number of applicative modules built on top of this environment, for the purpose of analyzing architectural drawings. A flexible user interface drives these modules. We also compare our choices with those of similar systems.

1 Introduction

Our research group has been investigating various aspects of graphics recognition techniques for more than ten years. In the last two years, we have also conducted a "consolidation" activity, especially for low-level graphics recognition methods [1], in order to build up a set of stable software components, reusable from one application to the other. This leads to a number of *systems engineering issues*, which we try to document in this paper.

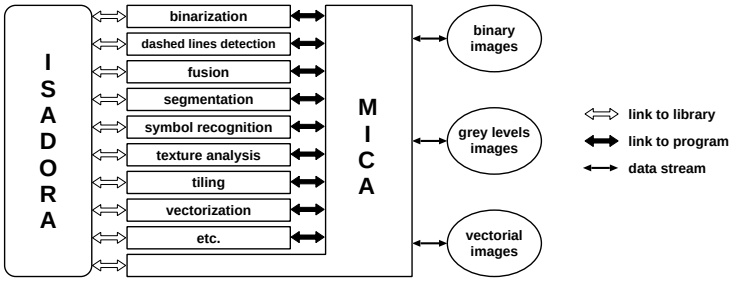


Figure 1: Overview of our three-layer system.

Fig. 1 gives an overview of our system, which includes three layers. The first layer is the ISADORA Library (§ 2), which consists of basic graphics recognition methods. They are designed to be as general as possible, *i.e.* to be

[a]This work is partially funded by France Telecom CNET.
[b]Also affiliated with Universitat Autònoma de Barcelona, Spain.

independent of application fields.

The second layer is an applicative layer, including useful graphics recognition *applications* (§ 3). Low-level applications are more or less sequences of calls to the corresponding methods of the ISADORA library, while higher-level ones are real programs including calls to ISADORA functionalities. All applications are independent programs, which can be either run using a command line or driven by the user interface.

The third layer is the user interface. In order for a document analysis system to work in practice, the user must be "in the loop". The interface must be tightly connected with the underlying layers, so that the user can easily and quickly guide the analysis process, and take corrective actions when necessary. The MICA user interface we propose (§ 4) provides such functionalities. It is also an application, linked with ISADORA like the others, but it is able to call applications of the second layer.

Before concluding the paper, we provide some comparisons of our work with other similar environments (§ 5).

2 Isadora: A Library of C++ Classes for Graphics Recognition

2.1 General Criteria and Choice of Tools

Before answering the specific needs of graphics recognition, common software engineering requirements must be taken into account. Firstly, we had to make the *transition from existing code* to new code as easy as possible. As we had a lot of old code written in C, C++ appeared to be the ideal choice, although it can be argued that it is not necessarily the best object-oriented language.

Secondly, we wanted to use as often as possible *public-domain tools and de-facto standards*. This explains, for example, why we use Jef Poskanzer's *Portable Bitmap* (PBM) format for image files, DXF for the representation of 2D graphics, and VRML for 3D graphics.

Finally, we needed to guarantee the *reusability* and the *efficiency* of the code. Reusability is achieved through C++ thanks to data abstraction and encapsulation, but it sometimes leads to a lot of computation overhead. We therefore allowed ourselves to use well-known "programming tricks", based on low-level C constructions, and hidden in so-called *private classes* (with private interfaces), which are not accessible to the common users of the library.

The next step was to choose the programming tools. This is out of the scope of this paper, but two points deserve a special attention. We had no intention to code a new set of common data structures like vectors, lists, sets, and so on. The C++ standard library provides most of them in what was previously known as the Standard Template Library (STL). A notable exception

is graphs, for which we therefore decided to use an additional library: LEDA[c], from the Max-Planck-Institut für Informatik in Saarbrücken, Germany.

Moreover, a software library is useless without documentation giving detailed information about class interfaces. We chose Malte Zöckler and Roland Wunderling's documentation system named DOC++[d], which is inspired by `javadoc`. It automatically generates online browsable HTML and high-quality hardcopy documentation directly from the C++ code by parsing the sources for special comments, that instruct how to create the documentation.

2.2 Design of the Classes

In fact, the fundamental problem we had to solve is encountered when implementing image processing operations in an object-oriented language. The object-oriented paradigm is based on data encapsulation, whereas image processing is generally seen as a collection of *operators* applied to images. By way of simple example, let us consider the convolution of an image by a Gaussian, that yields a new image $J = I \otimes G$. The question is then: Should convolution be defined as a function member of the `Image` class, or should it be defined as a global operator?

Our answer to this question is quite pragmatic, and is close to what is proposed by commercial libraries such as the *Image Vision Library*™ from Silicon Graphics. The basic idea is very simple. `Image` is the base class of a hierarchy. Its derived classes define image types: `BinaryImage`, `GreylevelImage`, `FloatImage`... For each of these classes, each image processing operation corresponds to a derived class and is implemented by a constructor of this derived class. Different methods to perform the same conceptual operation can thus be easily implemented through derived classes of an abstract class. Of course, this paradigm does not only apply to image processing, but also to all analysis and recognition tasks. It has the advantage of meeting the understandability requirements: Designers as well as clients of the library write compact and easy to read code.

2.3 Organization of the Class Hierarchy

ISADORA classes are hierarchically organised as a tree, with a single root named `IsaObject`, which contains information common to all objects, especially error handling facilities. This tree can be viewed as a collection of subtrees grouping together classes according to the different kinds of objects to be handled when

[c] `http://www.mpi-sb.mpg.de/LEDA/`
[d] `http://www.zib.de/Visual/software/doc++/`

designing a document analysis system. Each subtree is also organised in the same way.

There are three main families of classes. First, we have a number of *classes for image processing* (mainly `Mask`, `Histogram` and `Image` subtrees), as graphics processing involves a lot of image processing, at least in the low levels. Images can be either input data or results of some processing. They are represented as arrays of pixels. We intentionally do not use a too general definition, to avoid the complexity found for instance in the IUE specifications (§ 5). Most of the common image processing tools are available: Histogram computing, basic processing using convolutions (Gradient, Laplacian, etc.), mathematical morphology, edge detection, binary image processing, and so on.

Second, the `Graphics` subtree provides *classes for graphics processing*, including all the different kinds of graphical primitives, which are delivered by various segmentation modules, and also groupings of such primitives, which are delivered by some analysis module: Points, segments, chains of segments, rectangles, arcs of circle, connected components, etc. as well as ordered collections of such objects (§ 3.3).

Finally, we need *classes for utilities*, especially file processing (`IsaFile` subtree), to store graphics or image data in a selected format (§ 2.1).

3 The Application Layer

Let us now present some of the application modules we have designed, going from lower-level image processing tools to higher-level ones. The former are basically designed as a sequence of ISADORA objects, and can easily be reused for other graphics recognition tasks. The latter are more specifically oriented towards architectural drawing processing.

3.1 Binarization

When we need another binarization than that provided by commercial software or hardware, we have chosen the adaptive algorithm proposed by Trier and Taxt [2], with some minor adaptations [1]: Instead of using *ad-hoc* filters such as the Sobel gradient, as proposed by Trier and Taxt, we use Gaussian filtering, which has become standard in edge detection, and which happens to be implemented in a robust way in our library.

3.2 Segmentation

Most text/graphics separation methods are based on analyzing the connected components. For that purpose, we have designed the `LabelImage` class, whose

constructor builds a tree of connected components from a `BinaryImage`[3]. The tree represents the inclusion relation between components and gives the oriented contours of all the components.

The principle of the algorithm consists in analyzing the input image one row after the other, comparing the current row with the previous one, and storing all necessary information while labelling the black and white runs of the current row. The contours of the connected components are chained "on the fly" and described by Freeman chain codes, represented by the `Freeman` class. This class defines just one possible representation of a *chain*:

```
class Freeman : public GenChain<int>
```

While the template class `GenChain<T>` defines a common interface to all kinds of chains (§ 3.3). Hence, the `Freeman` class defines the specific encoding of Freeman chains, and implements the common interface given by `GenChain`.

One of the best text/graphics separation methods explained in literature is that of Fletcher and Kasturi[4]. In our implementation of this method, we added an absolute threshold for the size of a text component[1]. This is followed by string grouping, using the Hough transform, as proposed by Fletcher and Kasturi.

Further refinement of the graphics part can be obtained by separating thin and thick lines using morphological filtering, which is also available in the image processing part of the ISADORA library.

3.3 Vectorization

Among all the methods available for vectorization, *i.e.* raster-to-graphics conversion, our current favourite is *skeletonization* based on the 3–4 distance transform[5], followed by some polygonal approximation[1]. The distance skeleton is implemented by the `LabeledSkeleton` class.

Once a skeleton is computed, the skeleton pixels must be linked into *chains*. The `LinkedChainsList` class defines a list of linked chains of 2D points. We show here the context of this class, to illustrate how a specific implementation (such as using a list of points to represent a chain) can be encapsulated in a generic, abstract class.

First, the `GenChain` template class provides a generic interface for any chain, independently of the way it is internally represented and coded.

We can then build a possible implementation of such a chain, using a list of points provided with an iterator. Of course, the functions given in the generic interface must be defined, although we do not show these implementations here for the sake of brevity.

```
template <class T>
  class GenLinkedChain : public GenChain<T> {
protected:
  /// The list itself
  list< GenPoint<T> > theList;
  /// An iterator on the list
  list< GenPoint<T> >::iterator theIter;
public:
  // Implementation of interface defined by GenChain<T>
```

Finally, a complete chaining can be defined as a list of such chains, with integer coordinates:

```
class LinkedChainsList : public list< GenLinkedChain<int> >
  { ... }
```

This class is provided with both a general constructor from a `BinaryImage` object, which chains any kind of binary image, and a specific constructor from a `LabeledSkeleton` object, implementing a linking algorithm which takes into account the topological properties of skeletons.

The next step is *polygonal approximation*. The way we have designed our library allows us to have several algorithms in store for that. They can be easily tested, be it on a contour of a connected component represented by a Freeman code, on a contour computed from a grey-level image by some edge detector, or on a skeleton. All we need is to have a list of objects implementing the generic interface for chains.

All the polygonal approximation methods use the generic interface provided by the `GenChain<int>` class. We have implemented two methods: That of Wall and Danielsson [6] and the recursive method proposed by Rosin and West [7], which is actually an evolution of an algorithm first proposed by David Lowe. Both classes, `WallDanielssonSegList` and `RosinWestSegList`, have a constructor from a `GenChain<int>` object, which means that they work on *any* kind of linked chain.

If necessary, vectorization can be in the same way followed by arc detection and dashed-line detection. The classes implementing these steps are designed according to the same principles.

3.4 Higher-level entities: Symbols and Textures

To identify symbols representing building elements, such as doors or windows, we have designed a flexible symbol recognition system, where the set of models is described as a network of constraints on graphical features [8].

We also need to recognize architectural textures, which are usually structures composed of similar lines presenting a regular repetition. The method we have integrated in our system results from previous work by one of the authors, at *Universitat Autònoma de Barcelona* [9]. The basic idea is to group similar neighbouring texels using a hierarchical clustering method.

4 Mica: The User Interface

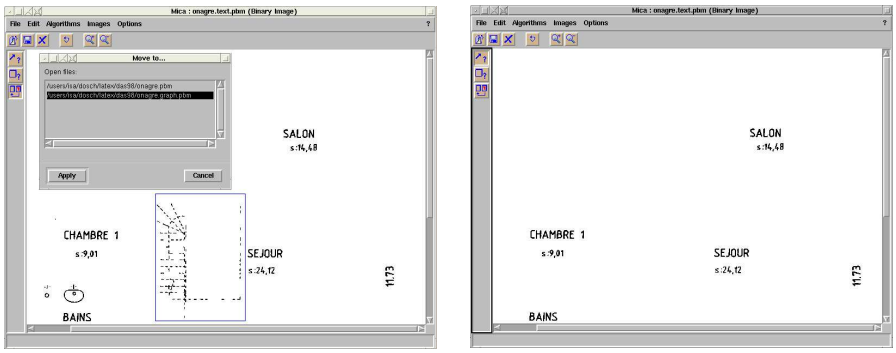
All the processes presented in the previous sections are incorporated into a user-friendly software system, provided with a sophisticated user interface giving a visual feedback about the working of the system. A user can guide the analysis process by setting values to parameters and thresholds of the different applications of the analysis. He can also perform several applications related to a given problem, in order to be able to select the one giving the best results.

In fact, it would not be reasonable to run the whole interpretation chain without giving the user any possibility to interact. Indeed, we have to deal with many problems like the noise introduced by artefacts (like folds) in drawings and by the methods sequentially applied, the fact that a same architectural component (door, window, etc.) can be drawn in many different ways, and, of course, the very limitations of our methods. Anyway, a human assistance is required to determine when and how a specific process is to be performed.

All the applications of the second layer are therefore connected through simple links with what constitutes the third layer of our analysis system, called MICA (Fig. 1). In this way, an application can be easily substituted for another or can be upgraded when necessary. Each link is used to transmit the values of the parameters to the corresponding application: Names of images to process, options, thresholds, etc. They are set with default values which can be customised by the user.

The basic functionalities of the MICA interface are:

- Display the contain of all kinds of files which are handled by the analysis, with common editing functionalities like multi-file editing, zooming, and so on.
- Tune parameter and threshold values, after examination of the results of the current application using the display facility.
- Directly manipulate resulting data, *i.e.* add missing results, delete or alter erroneous results. In particular, special editing operations are supported for both bitmap and vectorial images: Cut and copy of bitmap images (Fig. 2), creation and modification of components of a vectorial image, etc.



(a) Interactive correction of segmentation results.

(b) Final text layer.

Figure 2: Correction of text/graphics segmentation: Some dashes are misinterpreted as hyphens in the text layer (they actually represent stairs) and are moved to the graphics layer by the user.

5 Comparison with other Work

We are aware that we are not the only team working in this direction. For instance, the duality between object oriented programming and operator-based image processing (§ 2.2) has been studied by several teams, most notably in our area. Dov Dori's group, for instance, proposes a similar environment, the *Machine Drawing Understanding System* (MDUS)[10], which is based on the object-process methodology. Our methodological approach is probably more pragmatic than theirs, but the resulting environments are quite comparable.

Another system close to ours is TABS [11], also implemented in C++, with a flexible user interface built with TCL/TK. Whereas their applications deal with form processing and handwriting recognition, the design philosophy is similar to our choices, although TABS adds a supervision level driven by a blackboard.

In our opinion, a system like Khoros belongs to a different category. Its main strength comes more from the flexibility of the user interface (based on the visual programming paradigm) and from the exhaustivity of the image processing library. But although it can be very good for teaching or for prototyping, it remains quite slow when it is used for real applications.

Ultimately, our work, as well as others', might end up becoming contri-

butions to the *Image Understanding Environment* (IUE) [12], which aims at providing a complete environment for all kinds of image understanding activities. But before joining this large effort, with a huge library of components, we preferred to master a lower level of complexity by concentrating on our own know-how.

6 Conclusion

In this paper, we have proposed a framework for designing reusable graphics recognition software components. It comprises a library of basic image and graphics processing operations, ISADORA, and a set of higher-level graphics recognition applications. We have shown how a user interface can be added on top of these two layers.

The principles we have initially chosen to design each separate method and tool allow our group to subsequently develop this environment all together, thus going from single-user programming to group development. Some software engineering problems remain open, but our platform can be considered as operational and we are now experienced enough to progressively solve them. Of course, our research work about graphics recognition is still simultaneously carried on. When methods become mature enough, they are integrated into the common environment, and become thus available to the whole group.

We are aware that ultimately, we should aim at integrating the best of our work—with that of others—into a more general architecture, such as the IUE. But before being able to "serve the community", we want to prove that we can "serve our own group", by building a stable environment including robust implementations of the most useful algorithms.

References

1. K. Tombre, C. Ah-Soon, Ph. Dosch, A. Habed, and G. Masini. Stable, Robust and Off-the-Shelf Methods for Graphics Recognition. In *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia)*, pages 406–408, August 1998.
2. Ø. Due Trier and T. Taxt. Improvement of "Integrated Function Algorithm" for Binarization of Document Images. *Pattern Recognition Letters*, 16:277–283, March 1995.
3. D. Antoine, S. Collin, and K. Tombre. Analysis of Technical Documents: The REDRAW System. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 385–402. Springer-Verlag, Berlin/Heidelberg, 1992.

4. L. A. Fletcher and R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on PAMI*, 10(6):910–918, 1988.
5. G. Sanniti di Baja. Well-Shaped, Stable, and Reversible Skeletons from the (3,4)-Distance Transform. *Journal of Visual Communication and Image Representation*, 5(1):107–115, March 1994.
6. K. Wall and P. Danielsson. A Fast Sequential Method for Polygonal Approximation of Digitized Curves. *Computer Vision, Graphics and Image Processing*, 28:220–227, 1984.
7. P. L. Rosin and G. A. West. Segmentation of Edges into Lines and Arcs. *Image and Vision Computing*, 7(2):109–114, May 1989.
8. C. Ah-Soon and K. Tombre. Network-Based Recognition of Architectural Symbols. In A. Amin, D. Dori, P. Pudil, and H. Freeman, editors, *Advances in Pattern Recognition (Proceedings of Joint IAPR Workshops SSPR'98 and SPR'98, Sydney, Australia)*, volume 1451 of *Lecture Notes in Computer Science*, pages 252–261, August 1998.
9. G. Sánchez, J. Lladós, and E. Martí. Segmentation and Analysis of Linial Texture in Planes. In *Proceedings of 7th Spanish National Symposium on Pattern Recognition and Image Analysis, Barcelona, Spain*, volume 1, pages 401–406, 1997.
10. D. Dori. Representing pattern recognition-embedded systems through object-process diagrams: the case of the machine drawing understanding system. *Pattern Recognition Letters*, 16:377–384, April 1995.
11. C. Cracknell and A. C. Downton. TABS: script-based software framework for research in image processing, analysis and understanding. *IEE Proc.-Vis. Image Signal Process.*, 145(3):194–202, June 1998.
12. C. Kohl and J. Mundy. The Development of the Image Understanding Environment. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Seattle, Washington (USA)*, 1994.